Michael Eng
Jervis Muindi
Tim Sun

COMS 4115 Project Proposal

Description of Language

The language we intend to design is an L-system based fractal drawing language. L-systems use a variant of formal grammars to specify and generate fractals and other iterative sequences. Our language will be translated from a set of rules to turtle graphics procedures in Java, which will in turn be directly used to display the iterative sequence specified on screen.

Thus, using our language, it will be very easy for a user to write simple L-system grammar-like code and have us do the processing and computation necessary to paint and draw it for them. In addition, we also plan to support basic programming primitives to as to enable our language to be used to implement basic algorithms such as the the Euclidean algorithm.

Ideally, we would like to minimize the amount of parsing required from a given program. As such, we'd like to have Java classes that provide drawing and computational functionality that would execute alongside the outputted code from our compiler. To clarify, the compiler would output instructions for the drawing class to construct a fractal, and these instructions would call methods from a pre-made Java drawing class.

How Language should be used

The language will be used to draw fractals and other iterative sequences. We intend to include conditional statements and flow control functionality by allowing users to embed Java code directly into their programs.

That said, our language will also support basic programming primitives and hence it can also be used as a basic general purpose programming language to implement algorithms such as the Euclidean algorithm to find the greatest common divisor.

An Interesting Representative Program

The following program draws an n-th order approximation to the Hilbert curve using an L-system, and is based on the L-system described in the Wikipedia article for the Hilbert curve. The Hilbert curve is a fractal and a space-filling curve.

```
hilbert(n):{
  alphabet: (A,B;f,r,l,s);
  rules:
    lambda -> s A;
    A -> l B f r A f A r f B l;
    B -> r A f l B f B l f B r;
    A = ;
    B = ;
    f = turtle forward(150/3^n);
    r = turtle rotate(-60);
    l = turtle rotate(60);
    s = turtle place(0,0);
}

display(hilbert(5));
```

Language Syntax examples

`alphabet`: is a keyword that lets users declare variable names to be used in their system.
`rules`: is a keyword that lets users declare rules to be used in their system.
`r = turtle rotate(-60);` users will be able to specify turtle graphics-style commands as rules, similar to the syntax used in Logo. Angle arguments will be interpreted as degree values instead of radians.
`display()` will allow users to specify a system that they've written to be drawn on-screen.
`#Comment` we will be using single line comments, prefaced by the # symbol.

Expression

An expression is either : a function call, variable declaration or an assignment operation. It represents a single complete instruction.

Boolean Expression

This is an expression that evaluates to either true or false.

Statements

A statement is made up of expressions and operators and it is terminated by a ";" at the end.

Control-Flow

The syntax for program control flow will be similar to that exists in C and C-derived languages and the specifics are given below:

```
if (<<boolean_expression>>) {
        //Program statement goes here
} else {
        //Program statement goes here
}
```

If the boolean expression is true, then code in the subsequent block is executed and if it's false code in the else block is executed instead.

```
while(<<boolean_expr>>{
```

}
For as long as the boolean expression is true, the code enclosed in the curly braces is executed.

Data types
We also support the basic data types listed below :

- int : this is an integer
- double : this is a floating point number with double precision
- string : this is a sequence of characters

Declarations
Before a variable can be used, it must be declared.
The assignment of a value to the variable can be done at the same time as the declaration.

```
<<variable type>> variable_name;

<<variable type> variable_name =  variable_value;
```

Concrete Examples
```
int x; # declare x.
int x = 1; #declare x and instantiate with a value of x.
```

NB : Unassigned variables (i.e. variables that have been declared but not yet assigned a value) do not have

Operators
Our language will support the following operators for dealing with floating point numbers and integers:
```
double a;
double b;
```

a + b : Addition
a - b : Subtraction
a * b : Multiplication
a / b : Division
a > b : Greater than
a < b : Less than
a == b : Equals
a >= b : Greater than or equal to
a <= b: Less than or equal to.